ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais

PCS 2059 - Inteligência Artificial

1a. Lista de Exercícios

Prof. Responsável: Jaime Simão Sichman

GABARITO

1. Em artigo publicado em 1950, Alan Turing apresentou à comunidade acadêmica o que ficou conhecido como teste de Turing, também chamado de jogo da imitação. Uma pessoa teclava perguntas para um ser humano e um computador, sem saber a qual deles estava se dirigindo. A ideia era diferenciar um do outro. Se não conseguisse identificar a máquina por suas respostas, a conclusão seria que a máquina podia pensar, e portanto havia passado no teste.

2.

Estado: situação (configuração) do ambiente no qual deve ser resolvido o problema.

Espaço de estados: conjunto dos possíveis estados.

Árvore de busca: estrutura de dados que representa o espaço de estados e as relações de sucessão entre eles.

Estado objetivo: estado no qual o problema é resolvido.

Função objetivo: função que, dado o problema, retorna o conjunto de estados objetivos.

Ação: decisão tomada dado que se está em determinado estado.

Função sucessor: função que, dado o estado corrente, retorna os pares (ação, sucessor), que representam os possíveis estados a que se pode chegar e as ações que devem ser tomadas para se chegar nesses estados.

Fator de ramificação: número máximo de estados sucessores de qualquer estado.

Problema das N rainhas: posicionar N rainhas em um tabuleiro de xadrez de forma que não haja possibilidade de ataque entre nenhuma delas (ou seja, não há duas ou mais rainhas na mesma linha vertical, horizontal ou diagonal)

Estado: configuração das rainhas no tabuleiro.

Espaço de estados: o conjunto de todas as disposições das rainhas no tabuleiro.

Estado objetivo: qualquer configuração em que as N rainhas estejam posicionadas no tabuleiro sem que haja possibilidade de ataque entre elas.

Função objetivo: conjunto de possíveis configurações em que as N rainhas estejam posicionadas no tabuleiro sem que haja possibilidade de ataque entre elas.

Ação: posicionamento de uma rainha em alguma posição vazia no tabuleiro ou reposicionamento de uma rainha que já se encontra no tabuleiro.

Função sucessor: função que, dado uma configuração do tabuleiro, retorna um conjunto de possíveis ações e as respectivas configurações resultantes.

Fator de ramificação: número máximo de possíveis estados sucessores de alguma configuração do tabuleiro.

3. Sejam:

- *b*: fator de ramificação
- *d*: profundidade do nó mais raso que representa um estado objetivo
- *m*: maior profundidade da árvore de busca

A busca em largura, por explorar todos os nós mais rasos antes de explorar os de profundidade maior, encontra o estado solução mais raso. Da mesma forma, caso o fator de ramificação *b* for finito, se existir uma solução ela sempre será encontrada, caracterizando uma busca completa. Já a busca em profundidade pode encontrar um solução em uma profundidade maior que *d*. Caso a maior profundidade da árvore de busca (*m*) for finita, o algoritmo encontrará uma solução. Porém, não é difícil encontrar exemplos de problemas em que *m* pode ser infinito. Qualquer problema em que uma ação pode ser "feita e desfeita", ou seja, em que um estado pode ser o sucessor de seu próprio sucessor, terá *m* infinito caso não seja possível guardar estados repetidos.

Quanto à complexidade espacial, a busca em profundidade se mostra mais eficiente do que a busca em largura. Enquanto a última tem que guardar todos os nós pertencentes à profundidade em que o algoritmo se encontra e mais todos os nós das profundidades anteriores (o que significa, no pior caso, todos os nós da profundidade $d+1$, caracterizando uma complexidade espacial $O(b^{d+1})$), o algoritmo de busca em profundidade tem que expandir apenas os nós que se encontram no caminho que está sendo explorado (o que, no pior caso, corresponde à profundidade máxima *m*, caracterizando a complexidade espacial de $O(bm)$). Já com relação à complexidade temporal, a busca em largura pode apresentar vantagem. Como no pior caso a busca em largura gera todos os nós até a profundidade $d+1$, e a busca em profundidade gera todos os nós até a profundidade *m*, as respectivas complexidades temporais são $O(b^{d+1})$ e $O(b^m)$. Porém, *m* pode ser bem maior do que *d*.

4. Para resolver este problema, deve ser adotada uma ordem em que os nós são expandidos, ou seja, deve-se definir qual a ordem das peças a serem testadas. Aqui escolheu-se testar primeiro o movimento da peça mais à esquerda, depois a seguinte, e assim por diante.

Nas figuras de resolução do problema, os nós são representados por retângulos, dentro dos quais se encontra a informação da disposição das fichas: B(x,y);P(w,z) significa que as posições "x" e "y" estão ocupadas por fichas brancas e as posições "w" e "z" estão ocupadas por fichas pretas. O número das posições são ordenados em ordem crescente da esquerda para a direita (a posição 1 é a da extrema esquerda, e assim por diante). A letra "R" próximo a determinados nós significa que eles não foram expandidos por serem repetidos, sendo assim descartados.

a) Busca em largura (está na página seguinte)

b) Busca em profundidade (está na outra página)

c) Caso não seja possível guardar estados repetidos para eliminá-los da árvore de busca, para este problema apenas a busca em largura seria adequada. Como é possível movimentos de ida e volta das fichas, a busca em profundidade provavelmente se perderia expandindo um caminho com profundidade infinita, tentando mover uma peça para um lado, e logo em seguida para o outro.

Busca em profundidade:

B(1,2);P(4,5)

B(2,3);P(4,5) B(1,3);P(4,5) B(1,2);P(3,5) B(1,2);P(3,4)

B(1,3);P(4,5) B(1,2);P(4,5)

B(2,3);P(4,5) R B(1,2);P(4,5) R B(1,3);P(2,5)

B(1,3);P(4,5) R B(1,4);P(2,5) B(1,3);P(2,4)

B(3,4);P(2,5) B(1,4);P(3,5) B(1,3);P(2,5) B(1,4);P(2,3)

B(3,4);P(1,5) B(1,4);P(2,5)

B(3,4);P(2,5) R B(2,4);P(1,5) B(2,3);P(1,5)

B(2,4);P(3,5) B(3,4);P(1,5) R B(2,3);P(1,5) B(2,4);P(1,3)

B(1,4);P(3,5) B(2,4);P(1,5) R B(3,4);P(1,5) R B(2,4);P(1,5) R B(2,3);P(1,4)

B(2,4);P(3,5) R B(1,4);P(2,5) R B(1,2);P(3,5) B(2,5);P(1,4) B(2,3);P(1,5)

B(1,4);P(3,5) R B(1,2);P(4,5) R B(1,2);P(3,4) B(2,5);P(3,4) B(3,5);P(1,5) B(2,5);P(1,4) B(2,3);P(1,4)

SOLUÇAO

B(1,2);P(4,5) R B(1,2);P(3,5) R

Descartado

5. Uma possível estratégia de busca é utilizar a busca com profundidade interativa, ordenando os *links* de cada página de acordo com o quanto se avança em termos de nome de domínio na Web. Assim, enquanto se está em uma página com nome de domínio diferente da página objetivo, os *links* para URLs com nomes de domínio diferentes da página do momento são expandidos primeiro. Quando se chega a uma página com nome de domínio igual à página objetivo, expande-se primeiro os *links* com mesmo nome de domínio.

A busca bidirecional não parece ser uma boa alternativa, pois requereria a implementação da função predecessor, indicando todas as páginas que fazem *link* com a página de interesse. O número de predecessores de uma página pode ser algo muito grande, e dificilmente serão computadas todos os predecessores. Um mecanismo de busca na Web, como o Google, até poderia ser utilizado, mas as heurísticas e podas utilizadas nesses mecanismos podem "comer" os predecessores de interesse. Além disso, o tempo de computação deveria incluir o tempo gasto por esses mecanismos para executar a busca em seu banco de dados.

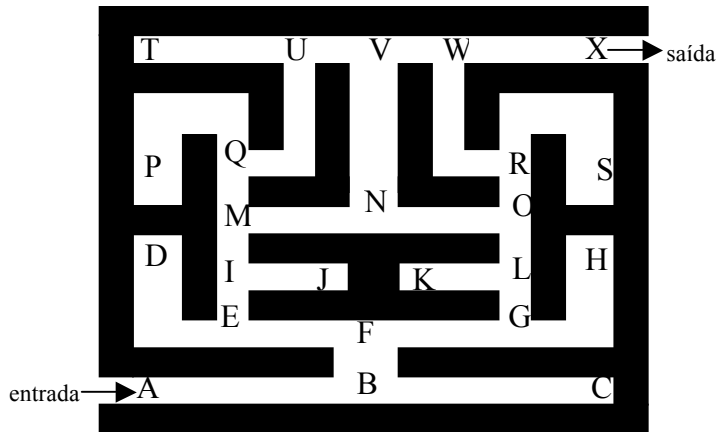
6. Para a resolução deste exercício, foram adotadas as seguintes podas das árvores:

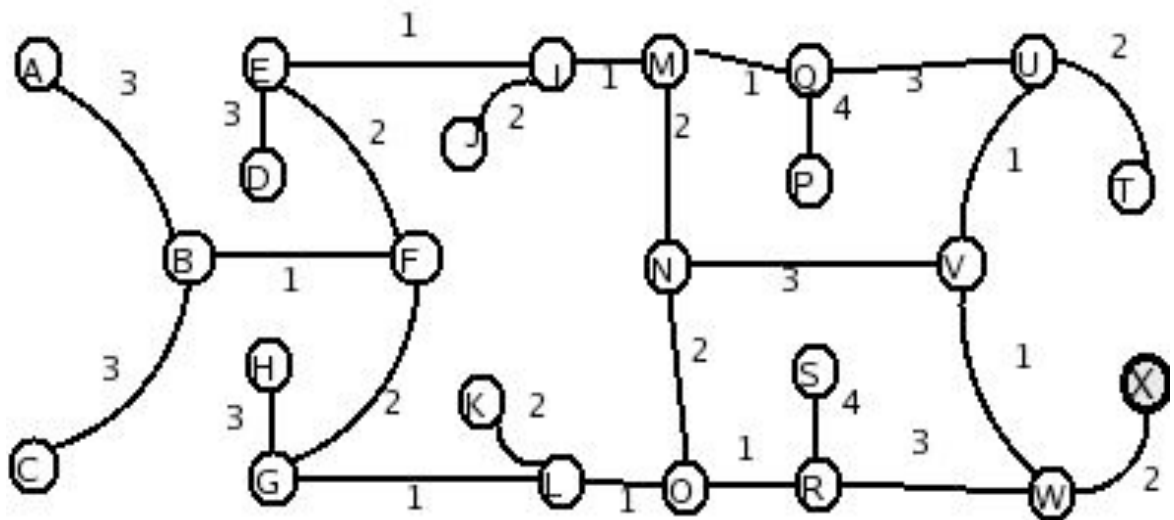
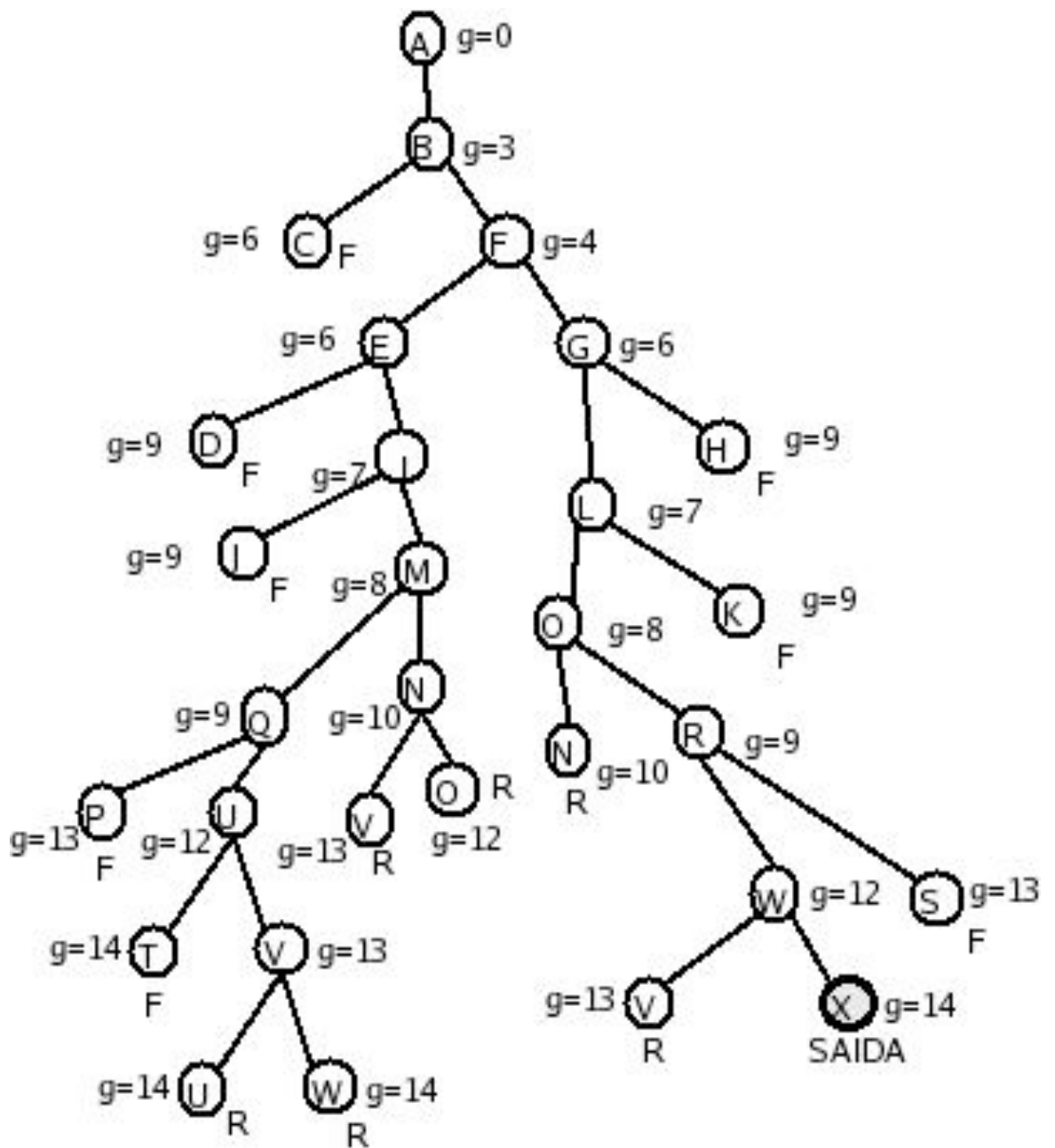
- Quando um nó é sucessor de um sucessor, ele não é expandido. Isso quer dizer que, indo de A para B, o nó referente a A é desconsiderado quando da expansão do nó referente a B. Isso é possível porque nunca o melhor caminho em um mapa ou um labirinto faz um vai-e-volta. Como o busca por custo uniforme e o A* são ótimos, essas poda elimina muitos nós desnecessários, muito embora isso ***não faça parte do algoritmo***. O algoritmo original expande todos os sucessores, mesmo que um deles seja o próprio antecessor.
- Quando um nó que já foi expandido reaparece na árvore em outro caminho, ele só é expandido se tiver um custo de caminho *(g)* menor do que o da vez anterior. Caso contrário, ele não é expandido.

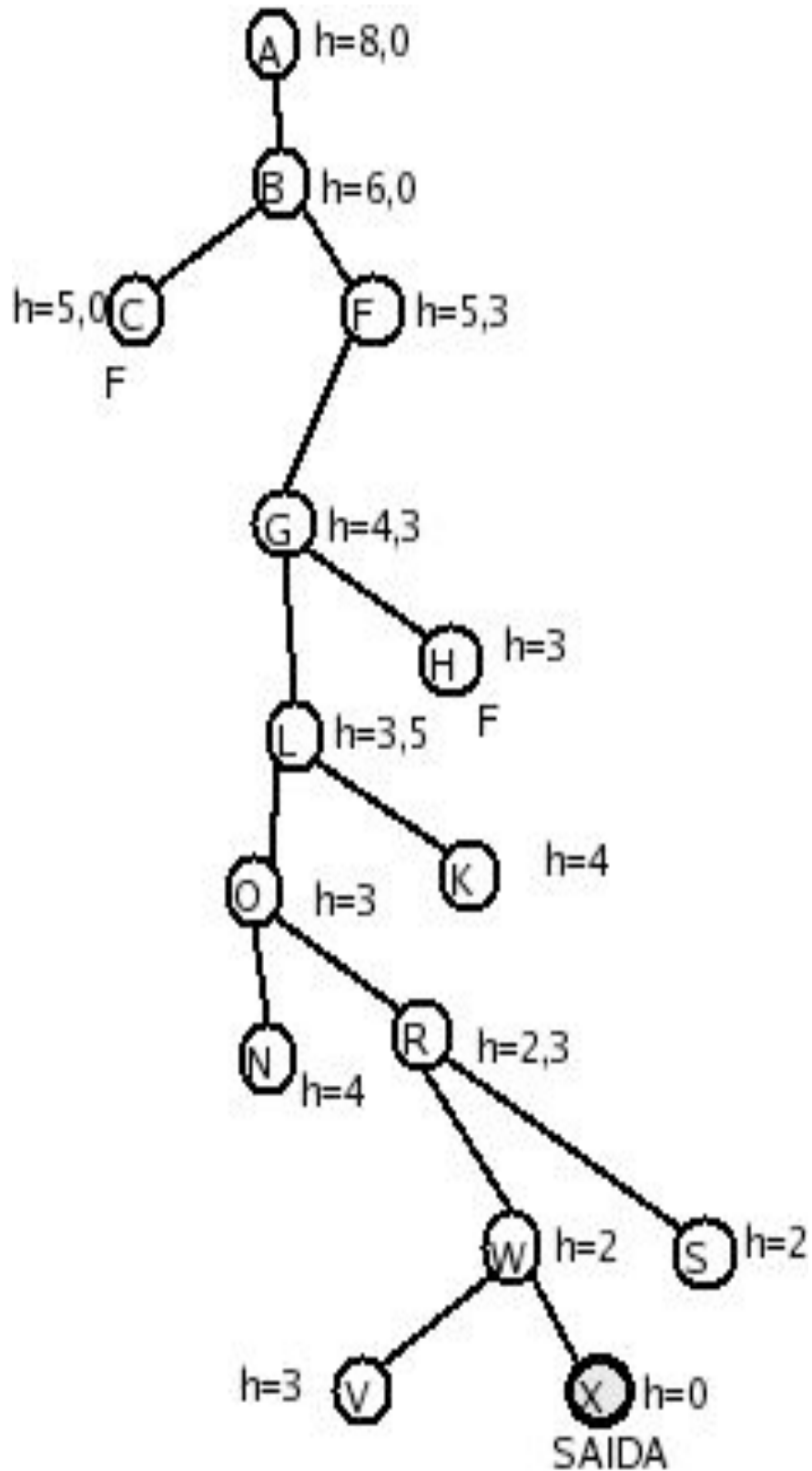
As letras F próximas aos nó árvores significam que ele seria expandido caso não correspondesse a um beco sem saída (cai no caso de que seu único sucessor seria o próprio predecessor). As letras R próximas aos nó árvores significam que ele seria expandido caso não correspondesse a um nó repetido na árvore e com custo de caminho maior ou igual ao da aparição anterior.

a) Construir o grafo, indicando as distâncias

Sejam os vértices do grafo nomeados como abaixo:

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais

Então, o grafo fica:

b) Resolução por busca de custo uniforme

A g=0
B g=3
g=6 C F
F g=4
g=6 E
G g=6
g=9 D F
I g=7
H F g=9
L g=7
g=9 J F
g=8 M
O g=8
K g=9 F
g=9 Q
g=10 N
N g=10 R
R g=9
P g=13 F
U g=12
g=13 V R
O R g=12
W g=12
S g=13 F
g=14 T F
V g=13
g=13 V R
X g=14 SAIDA
g=14 U R
W g=14 R

c) Resolução por busca pela melhor escolha

A h=8,0

B h=6,0

h=5,0 C F h=5,3

F

G h=4,3

H h=3

F

L h=3,5

K h=4

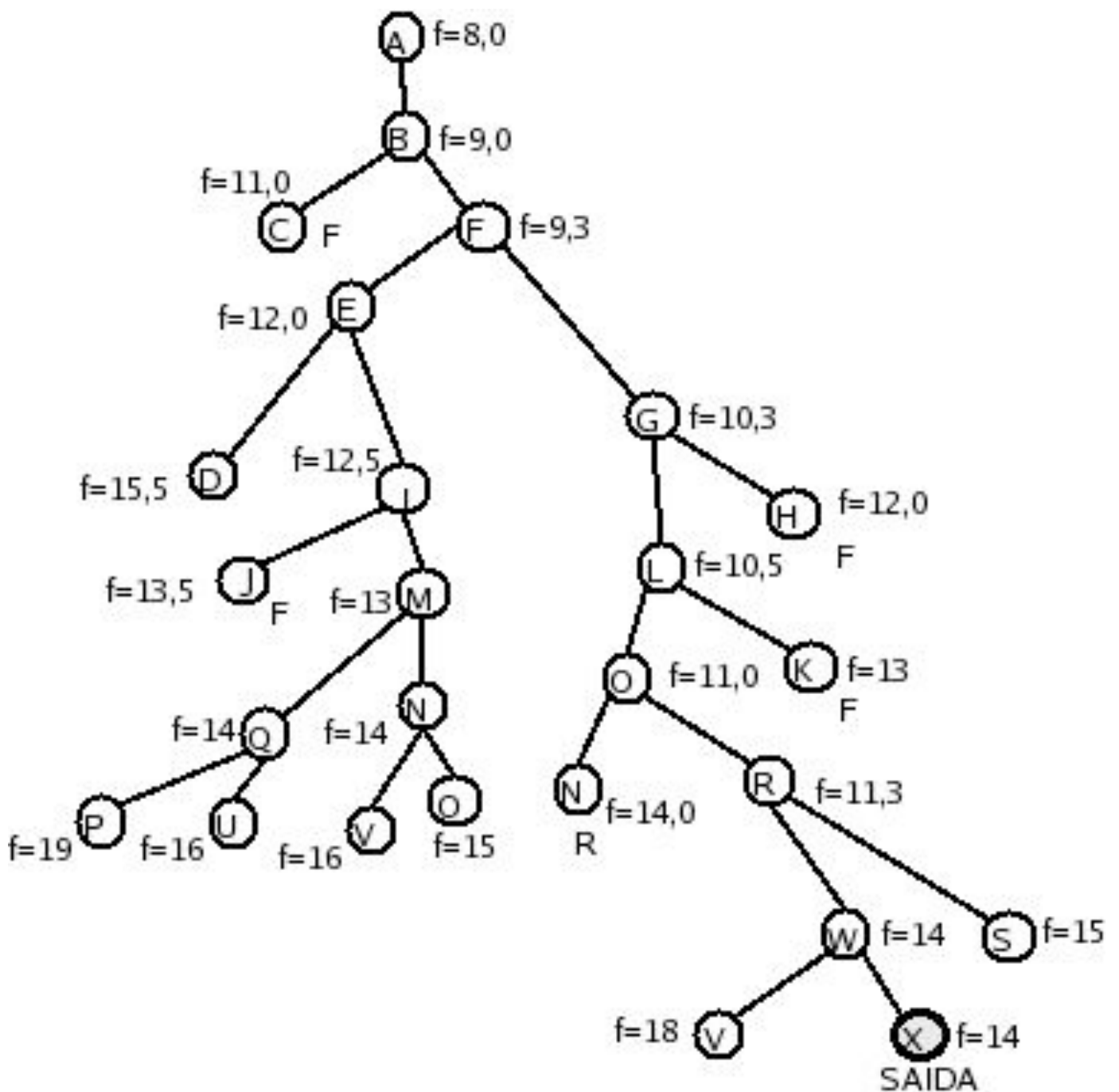
O h=3

N h=4 R h=2,3

W h=2 S h=2

h=3 V X h=0

SAIDA

d) Resolução por busca A*

A f=8,0
B f=9,0
f=11,0 C F
F f=9,3
f=12,0 E
G f=10,3
f=15,5 D
f=12,5 I
H f=12,0 F
f=13,5 J F
f=13 M
L f=10,5
O f=11,0
K f=13 F
f=14 Q
f=14 N
N f=14,0 R
R f=11,3
f=19 P
f=16 U
f=16 V
O f=15
W f=14
S f=15
f=18 V
X f=14 SAIDA

7. Nome dos algoritmos

a) A busca em feixe local com k=1 resulta em uma busca gulosa local ou uma subida da montanha pela encosta mais íngreme. O estado objetivo é sempre aquele com maior (ou menor) valor da função heurística.

b) A busca em feixe local com k=inf. já não é mais "local". Assim, se tornará uma busca cega que eliminará estados repetidos.

c) A têmpera simulada com T=0 deixará de realizar exploração, passando a ser uma descida pela encosta (não necessariamente a mais íngreme).

d) Quando a população do algoritmo genético é 1, não há possibilidade de crossover o permutação, sendo possível apenas o mecanismo de mutação. Dessa forma, torna-se uma busca aleatória.

8. Modelagens do problema do caixeiro viajante:

A modelagem de um problema para solução por busca exige a definição dos estados, da função sucessor, da função objetivo, do custo de caminho e da determinação de um estado inicial.

Problema do caixeiro viajante: Dadas N cidades, todas ligadas entre si, encontrar o menor caminho que passe por cada uma exatamente uma vez, até se chegar novamente na cidade inicial.

a) Modelagem para uma técnica de busca cega

Estados: cidade em que o caixeiro se encontra mais o conjunto de cidades que ele percorreu, ordenado de acordo com a ordem de visitação.

Estado inicial: cidade inicial, sem ter entrado em nenhuma outra cidade.

Função sucessor: retorna o resultado da ação de se ir a alguma cidade ainda não visitada, acrescentando a cidade de saída na lista de cidades visitadas.

Função de objetivo: retorna os estados em que a cidade corrente é igual à cidade inicial, e em que a lista de cidades visitadas contém todas as cidades do mapa.

Custo de caminho: soma dos custos de passo, que são os custos de se ir de uma cidade a outra.

b) Modelagem para uma técnica de busca informada

Estados: cidade em que o caixeiro se encontra mais o conjunto de cidades que ele percorreu.

Estado inicial: cidade inicial, sem ter entrado em nenhuma outra cidade.

Função sucessor: retorna o resultado da ação de se ir a alguma cidade ainda não visitada, acrescentando a cidade de saída na lista de cidades visitadas.

Função de objetivo: retorna os estados em que a cidade corrente é igual à cidade inicial, e em que a lista de cidades visitadas contém todas as cidades do mapa.
Custo de caminho: soma dos custos de passo, que são os custos de se ir de uma cidade a outra.

Heurística: valor do menor custo de se ir da cidade de interesse até outra cidade ainda não visitada.

c) Modelagem para uma técnica de busca local

Estados: conjuntos ordenados de N cidades.
Estado inicial: um conjunto ordenado em que a primeira cidade é a cidade inicial.
Função sucessor: retorna o resultado da ação de se substituir uma cidade do conjunto ordenado por outra (a não ser a cidade inicial), ou de se permutar duas cidades quaisquer no conjunto ordenado de cidades (a não ser a cidade inicial).
Função de objetivo: retorna os estados em que todas as cidades da lista são diferentes entre si.
Custo de caminho: soma dos custos custos de se ir de uma cidade a outra no caminho.

Heurística: valor do maior custo de uma cidade a outra vezes o número de repetições na lista de cidades, ou o custo de caminho quando se está em um estado objetivo.

9. Formulação

Rectilinear floor-planning

Sejam:

- n Є N, o número de peças retangulares
- L Є N, a largura (frente) do piso
- A Є N, a altura (fundo) do piso
- *larp* : {1,2,...,n} → N, uma função que retorna a largura de cada uma das n peças retangulares
- *altp* : {1,2,...,n} → N, uma função que retorna a altura de cada uma das n peças retangulares

VARIÁVEIS

X1, X2, ..., Xi,...., Xn - posição x de cada uma das n peças no piso
Y1, Y2, ..., Yi,...., Yn - posição y de cada uma das n peças no piso

DOMÍNIOS

Xi Є {0,1,...,L}
Yi Є {0,1,...,A}
para todo $1 \leq i \leq n$

Considera-se que a posição Xi=0,Yi=0 seja o canto superior esquerdo do piso e que a posição Xi=L,Yi=A seja o canto inferior direito do piso.

RESTRIÇÕES

Xi + *larp*(i) ≤ A
Yi + *altp*(i) ≤ B
(Xi + *larp*(i) ≤ Xj) OU (Yi + *altp*(i) ≤ Yj)

para todo $1 \leq i \leq n$, $1 \leq j \leq n$

Class Scheduling

Sejam:

- nc Є N, o número de cursos
- np Є N, o número de professores
- nr Є N, o número de salas de aula
- nt Є N, o número de horários
- *teach* : {1, ..., np} → Pot({1, ..., nc}), uma função que retorna para cada um dos np professores um subconjunto dos nc cursos que o professor está apto a ministrar.

VARIÁVEIS

P1, P2, ..., Pi, ... Pnc – professor que ira ministrar cada um dos nc cursos oferecidos.
R1, R2, ...,Ri, ... Rnc – sala na qual ira ocorrer cada um dos nc cursos.
T1, T2, ..., Ti, ... Tnc – horário alocado a cada um dos nc cursos.

DOMÍNIOS

Pi Є {1,...,np}
Ri Є {1,...,nr}
Ti Є {1,...,nt}
para todo $1 \leq i \leq nc$

RESTRIÇÕES

i Є *teach*(Pi)
Ti = Tj ➔ (Ri ≠ Rj) AND (Pi ≠ Pj)

para todo 1 ≤ i ≤ nc, 1 ≤ j ≤ nc

10. Problema criptoaritmético

As soluções estão representadas em forma de árvore, exibidas como em um gerenciador de arquivos. Dessa forma, todos os sucessores de um nó estão representados embaixo e à direita desse nó.

- Backtracking:

```
|- F=0
.  |- O=0 falha! Valor já atribuido a outra variável.
.  |- O=1
.  .  |- R=0 falha! Valor já atribuido a outra variável.
.  .  |- R=1 falha! Valor já atribuido a outra variável.
.  .  |- R=2
.  .  .  |- T=0 falha! Valor já atribuido a outra variável.
.  .  .  |- T=1 falha! Valor já atribuido a outra variável.
.  .  .  |- T=2 falha! Valor já atribuido a outra variável.
.  .  .  |- T=3 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0
.  .  .  |- T=4 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0
.  .  .  |- T=5 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0
.  .  .  |- T=6 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0
.  .  .  |- T=7 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0
.  .  .  |- T=8 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0
.  .  .  |- T=9 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e X3=0 RETROCESSO ...
.  .  |- R=3 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=4 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=5 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=6 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=7 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=8 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=9 falha! Não satisfaz [O + O = R + X1], X1=0 RETROCESSO ...
.  |- O=2
.  .  |- R=0 falha! Valor já atribuido a outra variável.
.  .  |- R=1 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=2 falha! Valor já atribuido a outra variável.
```

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais


```
.  .  |- R=3 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=4
.  .  .  |- T=0 falha! Valor já atribuido a outra variável.
.  .  .  |- T=1
.  .  .  .  |- U=0 falha! Valor já atribuido a outra variável.
.  .  .  .  |- U=1 falha! Valor já atribuido a outra variável.
.  .  .  .  |- U=2 falha! Valor já atribuido a outra variável.
.  .  .  .  |- U=3 falha! Não satisfaz [X1 + W + W = U + 10*X2, X1=0 e X2=0
.  .  .  .  |- U=4 falha! Valor já atribuido a outra variável.
.  .  .  .  |- U=5 falha! Não satisfaz [X1 + W + W = U + 10*X2, X1=0 e X2=0
.  .  .  .  |- U=6
.  .  .  .  .  |- W=0 falha! Valor já atribuido a outra variável.
.  .  .  .  .  |- W=1 falha! Valor já atribuido a outra variável.
.  .  .  .  .  |- W=2 falha! Valor já atribuido a outra variável.
.  .  .  .  .  |- W=3 bingo!
F=0 O=2 R=4 T=1 U=6 W=3
```

- Forward checking

```
|- F=0  ==>  Dom O,R,T,U,W={1,2,3,4,5,6,7,8,9}, X3=0
.  |- O=1 ==> Dom T={}, Retrocesso imediato ...
.  |- O=2 ==> Dom T={1}, X2=0, Dom R={4}, X1=0, Dom U={6,8}, Dom W={3,4}
.  .  |- R=4 ==> Dom T={1}, X2=0, X1=0, Dom U={6}, Dom W={3}
.  .  .  |- T=1
.  .  .  .  |- U=6
.  .  .  .  .  |- W=3 bingo!
F=0 O=2 R=4 T=1 U=6 W=3
```

- Variável mais restritiva

```
Candidatas: O          envolvida em 2 restriçoes;
            F,R,T,U,W envolvida em 1 restrição
|- O=0
.  |- F=0 falha! Valor já atribuido a outra variável.
.  |- F=1
.  .  |- R=0 falha! Valor já atribuido a outra variável.
.  .  |- R=1 falha! Valor já atribuido a outra variável.
.  .  |- R=2 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=3 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=4 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=5 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=6 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=7 falha! Não satisfaz [O + O = R + X1], X1=0
.  .  |- R=8 falha! Não satisfaz [O + O = R + X1], X1=0
```

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

```
      .  .  |- R=9 falha! Não satisfaz [O + O = R + X1], X1=0 RETROCESSO ...
RETROCESSO ...
      |- O=1
      .  |- F=0
      .  .  |- R=0 falha! Valor já atribuido a outra variável.
      .  .  |- R=1 falha! Valor já atribuido a outra variável.
      .  .  |- R=2
      .  .  .  |- T=0 falha! Valor já atribuido a outra variável.
      .  .  .  |- T=1 falha! Valor já atribuido a outra variável.
      .  .  .  |- T=2 falha! Valor já atribuido a outra variável.
      .  .  .  |- T=3 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0
      .  .  .  |- T=4 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0
      .  .  .  |- T=5 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0
      .  .  .  |- T=6 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0
      .  .  .  |- T=7 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0
      .  .  .  |- T=8 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0
      .  .  .  |- T=9 falha! Não satisfaz [X2 + T + T = 0 + 10*X3], X2=0 e
X3=0 RETROCESSO ...
      .  .  |- R=3 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=4 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=5 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=6 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=7 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=8 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=9 falha! Não satisfaz [O + O = R + X1], X1=0 RETROCESSO ...
      .  |- F=1 falha! Valor já atribuido a outra variável. RETROCESSO ...
      |- O=2
      .  |- F=0
      .  .  |- R=0 falha! Valor já atribuido a outra variável.
      .  .  |- R=1 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=2 falha! Valor já atribuido a outra variável.
      .  .  |- R=3 falha! Não satisfaz [O + O = R + X1], X1=0
      .  .  |- R=4
      .  .  .  |- T=0 falha! Valor já atribuido a outra variável.
      .  .  .  |- T=1
      .  .  .  .  |- U=0 falha! Valor já atribuido a outra variável.
      .  .  .  .  |- U=1 falha! Valor já atribuido a outra variável.
      .  .  .  .  |- U=2 falha! Valor já atribuido a outra variável.
      .  .  .  .  |- U=3 falha! Não satisfaz [X1 + W + W = U + 10*X2, X1=0 e
X2=0
      .  .  .  .  |- U=4 falha! Valor já atribuido a outra variável.
      .  .  .  .  |- U=5 falha! Não satisfaz [X1 + W + W = U + 10*X2, X1=0 e
X2=0
```

```
.   .   .   .   .   |- U=6
.   .   .   .   .   .   |- W=0 falha! Valor já atribuido a outra variável.
.   .   .   .   .   .   |- W=1 falha! Valor já atribuido a outra variável.
.   .   .   .   .   .   |- W=2 falha! Valor já atribuido a outra variável.
.   .   .   .   .   .   |- W=3 bingo!
O=2 F=0 R=4 T=1 U=6 W=3
```

- Variável mais restringida

```
Candidatas TODAS
|- F=0  ==>  Dom O,R,T,U,W={1,2,3,4,5,6,7,8,9}, X3=0
   Candidatas O,R,T,U,W
.  |- O=1 ==> Dom T={},
   Candidata T; RETROCESSO imediato ...
.  |- O=2 ==> Dom T={1}, X2=0, Dom R={4}, X1=0, Dom U={6,8}, Dom
W={3,4}
   Candidatas R,T,U,W
.  .  |- R=4 ==> Dom T={1}, X2=0, X1=0, Dom U={6}, Dom W={3}
      Candidatas T,U,W
.  .  .  |- T=1
            Candidatas U,W
.  .  .  .  |- U=6
             Candidata W
.  .  .  .  .  |- W=3 bingo!
F=0 O=2 R=4 T=1 U=6 W=3
```

11. Seja U(*n*) a utilidade calculada por para o nó *n* da árvore de jogo, e SUC(*n*) os a função sucessor do jogo aplicada a esse nó. Suponhamos, primeiramente, que MIN atue de forma não-ótima em apenas uma jogada representada na árvore, sendo que o nó a_1 representa essa jogada. A jogada seguinte resultante é representada pelo nó a_2. Caso MIN jogasse de forma ótima nessa jogada, a jogada seguinte seria representada pelo nó *b*. Em todas as jogadas posteriores, os jogadores tomam decisões minimax. Dessa forma, temos

$$U(a_1) = U(a_2) > U(b) = \min(U \circ SUC(a_1))$$

Se MIN jogasse de forma ótima, teríamos:

$$U(a_1) = U(b)$$

Seja a_0 o nó representando a jogada anterior a a_1, a_{-1} o nó representando a jogada anterior a a_0, e assim por diante. Se os valores de utilidade calculados forem propagados para as jogadas anteriores a a_1 pelo algoritmo minimax, então, a utilidades calculada para a jogada anterior será:

$$U(a_0) = \max((U \circ SUC(a_0)) \geq U(a_1) > U(b)$$

Se MIN jogasse otimamente, ela seria

$$U(a_0) = \max((U \circ SUC(a_0)) \geq U(a_1) = U(b)$$

Assim, $U(a_0)$ nunca é menor do que se MIN fosse ótimo. Para o nó anterior, temos:

$$U(a_{-1}) = \min(U \circ SUC(a_{-1}))$$

$U(a_{-1})$ também não diminui com relação ao que se teria com MIN ótimo, pois, se y>w, min(x,y,z) ≥ min(x,w,z). E assim por diante.

12.

b) Aproximadamente 9! Árvores possíveis.

a,c,d,e)